Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 204 (2008) 99–110
www.elsevier.com/locate/entcs

On Term-Graph Rewrite Strategies[1]

Rachid Echahed[2]

CNRS, LIG Laboratory
46, avenue Felix Viallet
38000 Grenoble
France

Abstract

We tackle the problem of cyclic term-graph rewriting. We first revisit the classical algorithmic approach to term-graph rewriting by providing a definition of rewrite rules of the form $lhs \rightarrow rhs$ where the left-hand sides are term-graphs and the right-hand sides are sequences of *actions*. Such actions, which specify how to rewrite a term-graph in a stepwise manner, contribute to simplify substantially the definition of cyclic term-graph rewriting. Then we define a new class of term-graph rewrite systems which are confluent over the so-called admissible term-graphs. Finally, we provide an efficient rewrite strategy which contracts only needed redexes and give pointers to other results regarding optimal rewrite strategies of admissible term-graphs.

Keywords: Term-graphs, Graph Rewrite Systems, Rewrite Strategies.

1 Introduction

Graph transformation is a general framework which is having various applications in computer science [26,13,14]. In this talk, term-graph rewriting will be considered as the underlying operational semantics of rule-based (functional or logic) programming languages (e.g. [24,9,3,23]). We will focus on cyclic term-graphs transformation. For acyclic term-graph rewriting, the reader may consult [25]. There are many reasons that motivate the use of (cyclic) term-graphs. They actually facilitate sharing of subexpressions which lead to efficient computations. They also allow one to go beyond the processing of first-order terms by handling efficiently real-world data-structures represented as cyclic graphs such as doubly-linked lists or circular lists, e.g. [8].

Computing with a term-graph rewrite system (tGRS) is not an easy task in general. Indeed, the classical properties of term rewriting systems (TRS) [27] can-

[1] This work has been partly funded by the project ARROWS of the French *Agence Nationale de la Recherche*.

[2] Email:Rachid.Echahed@imag.fr

doi:10.1016/j.entcs.2008.03.056

not be lifted without any caution to tGRSs. One of these properties is confluence. Consider for example the rule $F(a, a, x) \rightarrow x$ where a is a constant and x is a variable. This rule, which constitutes an orthogonal TRS, generates a confluent rewrite relation over (finite or infinite) terms whereas it generates a non confluent rewrite relation over term-graphs (see, e.g., [17]). It is well-known that this source of non-confluence of tGRSs comes from the so-called "collapsing rules" in orthogonal tGRSs. A rewrite rule is collapsing if its right-hand side is a variable. However, collapsing rules are very often used in programming and thus cannot be prohibited in any programming discipline. Most of access functions are defined by means of collapsing rules such as :
$car(cons(x, u)) \rightarrow x$
$cdr(cons(x, u)) \rightarrow u$
$left - tree(binTree(l, x, r)) \rightarrow l$
$right - tree(binTree(l, x, r)) \rightarrow r$

In practice, many programming languages are constructor-based, i.e., operators called *constructors*, which are intended to construct data structures are distinguished from operators called *defined operators* which are defined by means of rewrite rules. In this paper, we follow this discipline and consider orthogonal constructor-based tGRSs. We first revisit the classical algorithmic approach to term-graph rewriting [5] by providing a definition of rewrite rules of the form $lhs \rightarrow rhs$ where the left-hand sides are term-graphs and the right-hand sides are sequences of *actions*. Such actions, which specify how to rewrite a term-graph in a stepwise manner, contribute to simplify substantially the definition of cyclic term-graph rewrite process.

Then, we investigate the rewrite relation over a particular class of term-graphs called *admissible*. An admissible term-graph is a term-graph whose cycles do not include defined functions. We give a sufficient (syntactic) condition which ensures that the set of admissible graphs is closed under rewriting and we show the confluence of admissible graph rewriting relation, even in the presence of collapsing rules.

The confluence of a rewrite relation allows one to evaluate expressions in a deterministic and efficient way by using rewrite strategies. Such strategies have been well investigated in the setting of finite and infinite orthogonal TRSs (e.g., [22,16,18]). In [1], a strategy that computes outermost needed redexes based on definitional trees has been designed in the framework of orthogonal constructor-based TRSs. We show how definitional trees can be useful to design an efficient strategy in presence of orthogonal constructor-based tGRSs. We particularly state that the resulting strategy is c-hyper-normalizing on the class of admissible graphs and develops shortest derivations.

This extended abstract is organized as follows. In the following section, we revisit the definition of term-graph rewrite systems. Section 3 introduces the class of admissible term-graphs and state the confluence property for admissible tGRSs. An efficient rewrite strategy is presented in Section 4. Concluding remarks are given

in Section 5.

2 Term-Graph Rewrite Systems

Term-graphs [5] extend first-order terms by allowing sharing and cycles. There are many ways to define term-graphs rewriting in the literature. We can distinguish two broad approaches : the algorithmic approaches (e.g., [5]) and the categorical approaches (e.g., [7,15]). In this section we define a class of term-graph rewrite systems, denoted tGRS. We define the shape of its rules as well as the rewriting process. We follow and revisit the algorithmic approach. Our rules are close to those used in the language Lean [6] augmented by local redirection of pointers. The categorical definition of tGRSs following the double-pushout approach can be found in [8].

We start by giving some preliminary technical definitions.

Definition 2.1 *A* many-sorted signature $\Sigma = \langle S, \Omega \rangle$ *consists of a set* S *of sorts and an* S*-indexed family of sets of operation symbols* $\Omega = \uplus_{s \in S} \Omega_s$ *with* $\Omega_s = \uplus_{w \in S^*} \Omega_{w \to s}$. *We shall write* $f : s_1 \dots s_n \to s$ *whenever* $f \in \Omega_{s_1 \dots s_n \to s}$ *and say that* f *is of* sort s *and* rank $s_1 \dots s_n$.

A term-graph is defined in this paper as a set of nodes and edges between the nodes. Each node may be labeled with an operation symbol or not. A node which is not labeled will act as a variable. Let $\mathcal{N} = \uplus_{s \in S} \mathcal{N}_s$, be an S-indexed family of countable sets of *nodes*. $\mathcal{N}$ is supposed to be fixed throughout the rest of the paper.

Definition 2.2 *(Term-Graph)*
A term-graph g *over* $\langle \Sigma, \mathcal{N} \rangle$ *is a tuple* $g = \langle \mathcal{N}_g, \mathcal{N}_g^\Omega, \mathcal{L}_g, \mathcal{S}_g \rangle$ *such that :*

(i) $\mathcal{N}_g$ *is the set of nodes of* g, *i.e.,* $\mathcal{N}_g = \uplus_{s \in S} (\mathcal{N}_g)_s$ *with* $(\mathcal{N}_g)_s \subseteq \mathcal{N}_s$.

(ii) $\mathcal{N}_g^\Omega$ *is the subset of labeled nodes of* g, $\mathcal{N}_g^\Omega \subseteq \mathcal{N}_g$

(iii) $\mathcal{L}_g$, *the* labeling function of g, *is an* S*-indexed family of functions associating an operation symbol to each labeled node of* g, *i.e.,* $\mathcal{L}_g = \uplus_{s \in S} (\mathcal{L}_g)_s$ *with* $(\mathcal{L}_g)_s : (\mathcal{N}_g^\Omega)_s \to \Omega_s$.

(iv) $\mathcal{S}_g$, *the* successor function of g, *is an* S*-indexed family of functions associating a (possibly empty) string of nodes to each labeled node of* g, *i.e.,* $\mathcal{S}_g = \uplus_{s \in S} (\mathcal{S}_g)_s$ *with* $(\mathcal{S}_g)_s : (\mathcal{N}_g^\Omega)_s \to \mathcal{N}_g^*$ *such that for every node* $n \in (\mathcal{N}_g)_s$ *:*
- *if* $(\mathcal{L}_g)_s(n) = f$ *with* $f : s_1 \dots s_k \to s$, *then there exist* $n_1, \dots, n_k \in \mathcal{N}_g$ *such that* $(\mathcal{S}_g)_s(n) = n_1 \dots n_k$ *and* $n_i \in (\mathcal{N}_g)_{s_i}$ *for all* $i \in 1..k$.
- *if* $(\mathcal{L}_g)_s(n) = c$ *with* $c \in \Omega_{\varepsilon,s}$ *(*c *is a constant), then* $(\mathcal{S}_g)_s(n) = \varepsilon$ *(i.e.,* n *has no successor). We write* $n \in \mathcal{S}_g(m)$ *if* n *is a successor of* m.

We write $\mathrm{ar}(n)$ *the arity of node* n *which is equal to the length of* $\mathcal{S}_g(n)$. *A rooted term-graph, noted* g^n, *is a term-graph* g *with a distinguished node* n *(*$n \in \mathcal{N}_g$*) called the root of* g. n *will be noted* $\mathcal{R}oot_g$. *Let* g *be a term-graph and* n *and* m *two nodes of* g *(*$n, m \in \mathcal{N}_g$*), we write* $n \rightsquigarrow_g m$ *iff* $m \in \mathcal{S}_g(n)$. *We will say that node* m *is reachable in* g *from node* n *iff* $n \overset{*}{\rightsquigarrow}_g m$. *A rooted term-graph* g^n *consists of and only of nodes reachable from the root* n.

In the sequel, we will assume that all formulae we are considering are well-sorted, and thus drop subscripts related to the many-sorted framework.

As the formal definition of term-graphs is not very convenient to write examples, we recall below the linear notation [5] of term-graphs. In the following grammar, the variable A (resp. n) ranges over the set Ω (resp. $\mathcal{N}$) :

$$\textsc{TermGraph} ::= \textsc{Node} \mid \textsc{Node} + \textsc{TermGraph}$$

$$\textsc{Node} ::= n{:}A(\textsc{Node},\ldots,\textsc{Node}) \mid n{:}\bullet \mid n$$

The root of a rooted term-graph defined by means of a linear expression is the first node of the expression. $n{:}\bullet$ means that node n is not labeled.

Definition 2.3 (Homomorphism) *Let g_1^n and g_2^m be two rooted term-graphs. A homomorphism h from g_1^n to g_2^m is a mapping $h : \mathcal{N}_{g_1^n} \to \mathcal{N}_{g_2^m}$ which preserves the root, the labeled nodes and the successor functions, i.e., $h(n) = m$, $h(\mathcal{N}^{\Omega}_{g_1^n}) \subseteq \mathcal{N}^{\Omega}_{g_2^m}$, and for each labeled node, p, in g_1^n, $\mathcal{L}_{g_2^m}(h(p)) = \mathcal{L}_{g_1^n}(p)$ and $\mathcal{S}_{g_2^m}(h(p)) = h^*(\mathcal{S}_{g_1^n}(p))$ where h^* denotes the extension of h to strings (of nodes) defined by $h^*(p_1 \ldots p_k) = h^*(p_1) \ldots h^*(p_k)$.*

In [5], a rewrite step consists of diffrents stages such as (i) the build phase, which consists mainly in adding, beside the term-graph to be reduced, an instance of the right-hand side of the considered rule, together with the right connections to already existing nodes (ii) redirection phase, which consists in performing possible required pointer redirections and (iii) garbage collection phase. Definition of stage (i) is too technical and requires like stage (ii) a particular care in handling the names of nodes. In order to simplify the definition of rewrite steps, we will incorporate the actions required to reduce a term-graph directly in the right-hand sides of the rules. These actions are defined below.

Definition 2.4 (Actions) *An action is one of the following forms. We omit to give sort constraints which are quite straightforward and thus assume that all constructions are well-sorted.*

- *a **node definition** or **node labeling** "$\alpha : f(\alpha_1, \ldots, \alpha_n)$" where $\alpha, \alpha_1, \ldots, \alpha_n$ are nodes and f is a label of rank $s_1, \ldots, s_n$. This means that α is labeled by f and $\alpha_1 \ldots \alpha_n$ are the succesor nodes of α ($\mathcal{S}(\alpha) = \alpha_1 \ldots \alpha_n$).*
- *an **edge redirection** or **local redirection** "$\alpha \gg_i \beta$" where α, β are nodes and $i \in \{1, \ldots, \mathrm{ar}(\mathcal{L}(\alpha))\}$. This is an edge redirection and means that the target of the "ith" edge outgoing α is redirected to point β.*
- *a **global redirection** "$\alpha \gg \beta$" where α and β are nodes. This means that all edges pointing α are redirected to point β.*

The result of applying an action a to a term-graph g is denoted by $a[g]$ and is defined as the following term-graph g':

- *If $a = \alpha : f(\alpha_1, \ldots, \alpha_n)$ then $\mathcal{N}_{g'} = \mathcal{N}_g \cup \{\alpha, \alpha_1, \ldots, \alpha_n\}$, $\mathcal{L}_{g'}(\alpha) = f$, $\mathcal{L}_{g'}(\beta) = \mathcal{L}_g(\beta)$ if $\beta \neq \alpha$, and $\mathcal{S}_{g'}(\alpha) = \alpha_1 \ldots \alpha_n$, $\mathcal{S}_{g'}(\beta) = \mathcal{S}_g(\beta)$ if $\beta \neq \alpha$. $\cup$ denotes classical union.*

- *If* $a = \alpha \gg_i \beta$ *then* $\mathcal{N}_{g'} = \mathcal{N}_g$, $\mathcal{L}_{g'} = \mathcal{L}_g$, *and if* $\mathcal{S}_g(\alpha) = \alpha_1 \ldots \alpha_i \ldots, \alpha_n$ *then* $\mathcal{S}_{g'}(\alpha) = \alpha_1 \ldots \alpha_{i-1}\beta\alpha_{i+1} \ldots, \alpha_n$ *and for any node* γ *we have* $\mathcal{S}_{g'}(\gamma) = \mathcal{S}_g(\gamma)$ *iff* $\gamma \neq \alpha$.
- *If* $a = \alpha \gg \beta$ *then* $\mathcal{N}_{g'} = \mathcal{N}_g$, $\mathcal{L}_{g'} = \mathcal{L}_g$ *and for all nodes* δ *such that* α *occurs in* $\mathcal{S}_g(\delta)$, *i.e.,* $\mathcal{S}_g(\delta) = \alpha_1 \ldots \alpha_i \ldots \alpha_n$ *then* $\mathcal{S}_{g'}(\delta) = \alpha_1 \ldots \alpha_{i-1}\beta\alpha_{i+1} \ldots \alpha_n$ *and for any node* γ *we have* $\mathcal{S}_{g'}(\gamma) = \mathcal{S}_g(\gamma)$ *iff* α *does not occur in* $\mathcal{S}_g(\gamma)$

The application of an action a *to a rooted term-graph* g^n *is a rooted term-graph* g'^m *such that* $g' = a[g]$ *and root* m *is defined as follows :*

- $m = n$ *if* a *is not of the form* $n \gg p$.
- $m = p$ *if* a *is of the form* $n \gg p$.

The application of a sequence of actions u to a (rooted) term-graph g is defined inductively as follows : $u[g] = g$ if u is the empty sequence and $u[g] = u'[a[g]]$ if $u = a.u'$ where . is the concatenation operation.

Example 2.5 Let $G = n : f(m_1, m_2, m_3)$. Let $H_1 = m_1 : h(m_1)[G]$. Then $H_1 = n : f(m_1 : h(m_1), m_2, m_3)$. Let $H_2 = n \gg_2 m_1[H_1]$. Then $H_2 = n : f(m_1 : h(m_1), m_1, m_3)$. Let $H_3 = m_1 \gg m_3[H_2]$. Then $H_3 = n : f(m_3, m_3, m_3) + m_1 : h(m_3)$.

Definition 2.6 (Node Constraint) *A* node constraint *is a (possibly empty) conjunction of disequations between nodes:* $\bigwedge_{i=1}^{n}(\alpha_i \not\equiv \beta_i)$. *A substitution* $\sigma : \mathcal{N} \to \mathcal{N}$ *is a* solution *of a constraint* $\phi = \bigwedge_{i=1}^{n}(\alpha_i \not\equiv \beta_i)$ *iff for any* $i \in [1..n]$, *we have* $\sigma(\alpha_i) \neq \sigma(\beta_i)$. *We denote by* $sol(\phi)$ *the set of solutions of* ϕ.

Definition 2.7 (Rule) *A* term-graph rewrite rule *is an expression of the form* $[l \mid c] \to r$ *where* r *is a sequence of actions,* c *is a constraint and* l *is a rooted term-graph s.t. for any node* α *occurring in* l, *we have* $\mathcal{R}oot_l \overset{*}{\rightsquigarrow}_l \alpha$ *(i.e. any node occurring in the left-hand side must be reachable from the root* $\mathcal{R}oot_l$. *A rule* ρ_2 *is said to be a variant of a rule* ρ_1 *iff* ρ_2 *is obtained from* ρ_1 *by (one-one) renaming all the nodes in* ρ_1. *A* term-graph rewrite system *(tGRS) is a set of term-graph rewrite rules.*

Notice that classical first-order term rewrite systems are tGRSs. A term rewrite rule $g \to d$ can be easily presented as a term-graph rewrite rule of the form $[g \mid c] \to r$ where the constraint c is empty (true) and r consists of actions which construct the term d followed by the action $\mathcal{R}\text{oot}_g \gg \mathcal{R}\text{oot}_d$.

Example 2.8 As noticed above, any term rewrite system can be seen as a term-graph rewrite system. Below, we give some examples which illustrate the use and the benefit of pointer redirections. We define four functions *length*, *reverse*, *eq* and *fib*. The first function computes the length of a circular list, *reverse* performs the so-called in-situ list reversal, *eq* defines an equality operator over natural numbers. *eq* yields true at once if its arguments are located at the same place. The function *fib* defines the classical Fibonacci function, but requires a linear number of additions (+).

Length of a circular list :

$r : length(p : \bullet) \to r : length'(p, p)$

$r : length'(p_1 : cons(n : \bullet, p_2 : \bullet), p_2) \rightarrow r : s(0)$
$[r : length'(p_1 : cons(n : \bullet, p_2 : \bullet), p_3 : \bullet) \mid p_2 \neq p_3] \rightarrow r : s(q); q : length'(p_2, p_3)$

In-situ list reversal :
$o : reverse(p : \bullet) \rightarrow o : reverse'(p, q : nil)$
$o : reverse'(p_1 : cons(n : \bullet, q : nil), p_2 : \bullet) \rightarrow p_1 \gg_2 p_2; o \gg p_1$
$o : reverse'(p_1 : cons(n : \bullet, p_2 : cons(m : \bullet, p_3 : \bullet), p_4 : \bullet)) \rightarrow p_1 \gg_2 p_4; o \gg_1 p_2; o \gg_2 p_1$

Equality over naturals :
$p : eq(n : \bullet, n) \rightarrow q : true; p \gg q$
$[p : eq(n : 0, m : 0) \mid n \neq m] \rightarrow q : true; p \gg q$
$[p : eq(n : succ(n' : \bullet), m : succ(m' : \bullet)) \mid n \neq m] \rightarrow p \gg_1 n'; p \gg_1 m'$
$p : eq(n : succ(n' : \bullet), m : 0) \rightarrow q : false; p \gg q$
$p : eq(n : 0, m : succ(m' : \bullet)) \rightarrow q : false; p \gg q$

Fibonacci function (numbers) :
$r : fib(p : 0) \rightarrow r : 0$
$r : fib(p : succc(0)) \rightarrow r \gg p$
$r : fib(p : succc(succ(n))) \rightarrow r : f(p, q : succ(0); u : 0)$
$r : f(p : succc(succ(0)), q, u) \rightarrow r : +(q, u)$
$r : f(p : succc(v : succ(succ(n))), q, u) \rightarrow w : +(q, u); r \gg_1 v; r \gg_2 w; r \gg_3 q$

Definition 2.9 (Matching) *Let $[l \mid c] \rightarrow r$ be a rewrite rule and g^n a rooted term-graph. We say that the left-hand side $[l \mid c]$ matches the term-graph g^n at node p, and noted $[l \mid c] \leq g^p$ iff p is reachable from n (i.e. $n \overset{*}{\rightsquigarrow}_g p$) and there exists a homomorphism, also called* matcher, *h from l to g^p, i.e. $h : \mathcal{N}_l \rightarrow \mathcal{N}_g$ such that $h(\mathcal{R}oot_l) = p$ and h is a solution of constraint c, i.e., $h \in sol(c)$*

Definition 2.10 (Rewrite Step) *Let ρ be the rewrite rule $[l \mid c] \rightarrow r$ and g^n be a rooted term-graph. We say that g^n rewrite to g_1^m at node p by using the rule ρ iff there exists a matcher $h : l \rightarrow g^p$ which is a solution of constraint c and $g_1^m = h(r)[g^n]$. We write $g^n \rightarrow_{[p,[l|c]\rightarrow r]} g_1^m$ or simply $g^n \rightarrow g_1^m$.*

One of the main advantages of the class of term-graph rewrite systems described above (see also [8] for a definition of the same class following the categorical double-push out approach) is the ability to define in a natural way, rule-based programs with good space and/or time complexities, thanks to pointer redirection capabilities. Unfortunately, the expressiveness of this class has a cost : unlike term rewrite system, the property of confluence is no more ensured for orthogonal tGRSs. Therefore, defining functions (i.e., computing unique normal forms) by means of tGRSs necessitates, in general, to endow reducible term-graphs with some control over reducible sub-expressions. Examples of such controls are the annotations introduced in [19] or priorities over the nodes of [12]. These controls are somewhat close in spirit to the sequential ordering of instructions in imperative programs. Another alternative to recover confluence of rewriting consists in considering a sub-class of

term-graphs. This is the subject of the next section.

3 Admissible Term-Graphs

Computing with general cyclic term-graphs is not such an easy task even in presence of orthogonal rewrite systems. In this section we introduce the class of *admissible term-graphs* [9] for which confluence results can be stated and efficient strategies can be designed. This class is inspired from the imperative style of programming where defined procedures and functions operate over data-structures built using particular constructors such as records, pointers etc. and where cyclic expressions such as $n : fact(n)$, $n : tail(n)$ or $n : +(n, n)$ are meaningless.

Admissible term-graphs are defined in the context of constructor-based signatures.

Definition 3.1 (Constructor-based Signature) *A* constructor-based signature Σ, *is a triple* $\Sigma = \langle S, \mathcal{C}, \mathcal{D}\rangle$ *such that* S *is a set of sorts,* $\mathcal{C}$ *is an* S*-indexed family of sets of* constructor symbols, $\mathcal{D}$ *is an* S*-indexed family of sets of* defined operations, *such that* $\mathcal{C} \cap \mathcal{D} = \emptyset$ *and* $\langle S, \mathcal{C} \uplus \mathcal{D}\rangle$ *is a signature.*

Definition 3.2 (admissible rooted term-graph [9,10]) *A rooted term-graph* g^n *is admissible iff for all nodes* m*, labeled by a defined operation (i.e.,* $\mathcal{L}_{g^n}(m) \in \mathcal{D}$*),* m *is not reachable from itself (i.e.,* m *does not belong to a cycle* $m \overset{+}{\leadsto}_g m$*).*

Definition 3.3 (admissible tGRS) *Let* $SP = \langle \Sigma, \mathcal{R}\rangle$ *be a constructor-based tGRS.* SP *is called admissible iff for all rules* $[l \mid c] \to r$ *in* $\mathcal{R}$ *the following conditions are satisfied*

(i) l *is an admissible term-graph such that only the root of* l *is labeled by a defined symbol (i.e.,* $\mathcal{L}_l(\mathcal{R}oot_l) \in \mathcal{D}$*) and the remaining labeled nodes are labeled by constructor symbols.*

(ii) *for all global (respectively, local) redirections of the form* $p \gg q$ *(respectively,* $p \gg_i q$ *for some* i*), occurring in the right-hand side* r*, we have* $p = Root_l$ *and* $q \neq Root_l$*.*

(iii) *for all actions of the form* $\alpha : f(\beta_1, \ldots, \beta_n)$*, for all* $i \in 1..n$*,* $\beta_i \neq Root_l$

(iv) *the set of actions of the form* $\alpha : f(\beta_1, \ldots, \beta_n)$*, appearing in* r*, do not construct a cycle consisting only of newly introduced nodes in* r *and including a node labeled with a defined operation. If we note* $\leadsto_r$ *the reachability over the new nodes introduced in* r*, this condition could be specified as : for all nodes,* α*, introduced in* r *and labeled by a defined operation,* $\alpha \overset{+}{\not\leadsto}_r \alpha$*.*

Example 3.4 Examples mentioned in Example 2.8 but the list reversal rules are admissible.

The following proposition states that the class of admissible term-graphs is closed under the rewrite relation induced by an admissible tGRS.

Proposition 3.5 (Closure of admissible term-graphs [9]) *Let* $SP = \langle \Sigma, \mathcal{R}\rangle$ *be an admissible tGRS and* g^n *an admissible rooted term-graph. If* g^n *rewrites to*

q^m via a rewrite rule in $\mathcal{R}$, then q^m is also an admissible rooted term-graph.

Proposition 3.6 (Confluence of weakly orthogonal admissible tGRS [10]) *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a weakly orthogonal admissible tGRS (i.e., if two rules $[l_1 \mid c_1] \rightarrow r_1$ and $[l_2 \mid c_2] \rightarrow r_2$ in $\mathcal{R}$ overlap, then the instantiated right-hand sides produce the same graph up to renaming of nodes). Then SP is confluent (up to renaming of nodes) over the set of admissible term-graphs. That is to say, for all rooted admissible term-graphs g_1^n, $g_2^{n'}$, g_3^m and $g_4^{m'}$ such that g_1^n and $g_2^{n'}$ are identical up to renaming of nodes ($g_1^n \sim g_2^{n'}$), $g_1 \xrightarrow{*} g_3^m$ and $g_2 \xrightarrow{*} g_4^{m'}$, there exist two admissible graphs g_5^o and $g_6^{o'}$ such that $g_3^m \xrightarrow{*} g_5^o$, $g_4^{m'} \xrightarrow{*} g_6^{o'}$ and $g_5^o \sim g_6^{o'}$.*

Thanks to the confluence of the rewrite relation over admissible term-graphs, stated above, we proposed efficient rewrite strategies which compute only needed derivations. This is the object of the next section.

4 Efficient Reduction of Admissible Term-graphs

Rewrite strategies are often used in order to reduce the search space generated by a rewrite relation. Several rewrite strategies have been proposed in the literature in the framework of term rewrite systems (see, e.g., [27,16,20]).

In this talk, we will present a sequential rewrite strategy designed to reduce cyclic admissible term-graphs. This strategy contracts only needed outermost redexes and is c-hypernormalizing. This strategy can also be extended to an optimal parallel outermost rewrite strategy. We first introduce some vocabulary.

Definition 4.1 *(Strategy)*
Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an admissible tGRS. A (sequential) graph rewriting strategy *is a partial function $\mathcal{S}$ which takes an admissible graph g and returns a pair (p, R) such that p is a node of g, R is a rewrite rule of $\mathcal{R}$ and g can be rewritten at node p with rule R. We write $g \rightarrow_{\mathcal{S}} g'$ and speak of an $\mathcal{S}$-*step *from g to g' whenever $\mathcal{S}(g) = (p, R)$ and $g \rightarrow_{[p,R]} g'$. $\xrightarrow{*}_{\mathcal{S}}$ denotes the reflexive and transitive closure of $\rightarrow_{\mathcal{S}}$ and we speak of an $\mathcal{S}$-*derivation *from g to g' whenever $g \xrightarrow{*}_{\mathcal{S}} g'$.*
A strategy $\mathcal{S}$ is c-normalizing *iff for all admissible graph g admitting a constructor normal form c, if $g \xrightarrow{*} c$, then there exists a constructor graph c' such that $g \xrightarrow{*}_{\mathcal{S}} c'$ and $c' \sim c$.*
A strategy $\mathcal{S}$ is c-hyper-normalizing *iff for all admissible graphs g admitting a constructor normal form c, any derivation D starting with g which uses infinitely many times $\mathcal{S}$-steps ends with a constructor normal form c' such that $c \sim c'$.*

Definition 4.2 (Constructor Path) *We will say that a node p is reachable from a node n_0 in a term-graph g through a constructor path iff there exists a path in g, say $n_0 \rightsquigarrow_g n_1 \rightsquigarrow_g \ldots \rightsquigarrow_g n_k \rightsquigarrow_g p$ such that, for all $i \in 0..k$, $\mathcal{L}_g(n_j)$ is a constructor symbol ($\in \mathcal{C}$).*

Definition 4.3 (needed node, outermost redex) *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an admissible tGRS, g_1^n and g_2^m two term-graphs and $B = g_1^n \xrightarrow{*} g_2^m$ a rewrite derivation. A node q labeled with a defined operation in g_1^n and reachable from the root n* is a

residual node by B *if* q *remains reachable from the root* m *in* g_2^m. *Then, we call* descendant of g_1^q *the rooted term-graph* g_2^q. *A node* q *in* g *is* needed *iff in every rewriting derivation from* g *to a constructor normal form, a descendant of* g^q *is rewritten at its root* q. *A node* q *labeled with a defined operation in* g^n *is an* outermost node *of* g^n *iff* $q = n$ *or* q *is reachable from* n *through a constructor path. A redex* u *rooted by* q *in* g^n *is an* outermost redex *iff* $q = n$ *or* q *is reachable from* n *through a path* $p_0 \rightsquigarrow_{g^n} p_1 \rightsquigarrow_{g^n} \ldots \rightsquigarrow_{g^n} p_k$ *such that* $p_0 = n$, $p_k = q$ *and* g^{p_i} *is not a redex for all* $i \in 0..(k-1)$.

Remark : The notions of outermost node and outermost redex are well-defined in the framework of admissible graphs : if p and q are two nodes of an admissible graph labeled with defined operations and such that there exists a path from p to q (i.e., p is *outer* than q), then, by definition of admissible graphs, there is no path from q to p.

Our strategy is based on the notion of definitional trees introduced by Antoy [1] and are defined below. Nevertheless, our definition is a bit different from the original one proposed in [1] or those used in the context of graph rewriting [3,9,10]. The main difference comes from the use of branch nodes. We actually introduce the possibility to have some sharing in the left-hand sides of the rules. For that purpose, we distinguish in the definition below between *position.branch* and *share.branch* nodes of a definitional tree. *position.branch* corresponds to the *branch* nodes in [1] whereas *share.branch* gives another possibility to specialize patterns according to their topological shapes.

Definition 4.4 (Definitional tree) *Let* $SP = \langle \Sigma, \mathcal{R} \rangle$ *be an admissible tGRS. A tree* $\mathcal{T}$ *is a* partial definitional tree, *or* pdt, *with pattern* $\pi \mid C$ *iff one of the following cases holds :*

- $\mathcal{T} = rule([\pi \mid C] \to r)$, *where* $[\pi \mid C] \to r$ *is a variant of a rule of* $\mathcal{R}$.
- $\mathcal{T} = position.branch([\pi \mid C], o, \mathcal{T}_1, \ldots, \mathcal{T}_k)$, *where* o *is not a labeled node of* π, o *is of sort* s, $c_1, \ldots, c_k$ $(k > 0)$ *are different constructors of the sort* s *and for all* $j \in 1..k$, $\mathcal{T}_j$ *is a pdt with pattern* $\pi[o \leftarrow p : c_j(o_1 : \bullet, \ldots, o_n : \bullet)]$, *such that* n *is the number of arguments of* c_j *and* $p, o_1, \ldots, o_n$ *are new nodes.*
- $\mathcal{T} = share.branch([\pi \mid C], \mathcal{T}_1, \mathcal{T}_2)$, *where* $\mathcal{T}_1$ *is a pdt with pattern* $[\pi \mid C \wedge n \neq m]$ *such that* n *and* m *are nodes occurring in* π *and the constraint* $n \neq m$ *does not occur in* C *and* $\mathcal{T}_2$ *is a pdt with pattern* $[\pi' \mid C]$ *such that* π' *is obtained from* π *by collapsing the two nodes* n *and* m *(and their successors). I.e.* π' *is obtained by encoding the constraint* $n \doteq m$ *into* π.

We write $pattern(\mathcal{T})$ *to denote the pattern argument of a pdt.*
A definitional tree $\mathcal{T}$ *of a defined operation* f *is a* finite *pdt with a pattern of the form* $[p : f(o_1 : \bullet, \ldots, o_n : \bullet) \mid true]$, *also denoted by* $p : f(o_1 : \bullet, \ldots, o_n : \bullet)$, *where* n *is the number of arguments of* f, $p, o_1, \ldots, o_n$ *are new nodes, and for every rule* $[l \mid C] \to r$ *of* $\mathcal{R}$, *with* l *of the form* $f(g_1, \ldots, g_n)$, *there exists a leaf* $rule([l' \mid C'] \to r')$ *of* $\mathcal{T}$ *such that* $[l' \mid C'] \to r'$ *is a variant of* $[l \mid C] \to r$. *An* inductively sequential *tGRS is an admissible tGRS such for every defined function,* f, *there exists a definitional*

tree of f.

Example 4.5 The reader may verify that the examples mentioned in Example 3.4 are all inductively sequential term-graph rewrite systems.

Definition 4.6 (A term-graph rewrite strategy Φ) *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an inductively sequential tGRS and g^n a rooted term-graph. Φ is the partial function defined by $\Phi(g^n) = \varphi(g^p, \mathcal{T}_f)$, where p is reachable from root n through a constructor path in g^n, p is labeled with a defined operation f and $\mathcal{T}_f$ is a definitional tree of f. Let g^n be a rooted term-graph such that $\mathcal{L}_{g^n}(n) \in \mathcal{D}$ (i.e. the root n is labeled with a defined operation) and $\mathcal{T}$ a pdt such that $pattern(\mathcal{T}) \leq g^n$. We define the partial function φ :*

$$\varphi(g^n, \mathcal{T}) = \begin{cases} (n, [\pi' \mid C'] \to r') & \text{if } \mathcal{T} = rule([\pi \mid C] \to r) \text{ and} \\ & \quad [\pi' \mid C'] \to r' \text{ is a variant of } [\pi \mid C] \to r \ ; \\ \varphi(g^n, \mathcal{T}_i) & \text{if } \mathcal{T} = share.branch([\pi \mid C], \mathcal{T}_1, \mathcal{T}_2) \text{ and} \\ & \quad pattern(\mathcal{T}_i) \leq g^n \text{ for some } i \in 1..2 \ ; \\ \varphi(g^n, \mathcal{T}_i) & \text{if } \mathcal{T} = position.branch([\pi \mid C], o, \mathcal{T}_1, \ldots, \mathcal{T}_k) \text{ and} \\ & \quad pattern(\mathcal{T}_i) \leq g^n \text{ for some } i \in 1..k \ ; \\ (p, R) & \text{if } \mathcal{T} = position.branch([\pi \mid C], o, \mathcal{T}_1, \ldots, \mathcal{T}_k), \\ & \quad [\pi \mid C] \text{ matches } g^n \text{ at the root } n \text{ by} \\ & \quad \text{homomorphism } h : \pi \to g, \\ & \quad h(o) \text{ is labeled with a defined operation } f \text{ (in } g), \\ & \quad \mathcal{T}' \text{ is a definitional tree of } f \text{ and} \\ & \quad \varphi(g^{h(o)}, \mathcal{T}') = (p, R). \end{cases}$$

Proposition 4.7 *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an inductively sequential tGRS, f a defined operation, $\mathcal{T}_f$ a definitional tree of f, and g^n a rooted term-graph whose root is labeled with f (i.e. $\mathcal{L}_{g^n}(n) = f$). If $\varphi(g, \mathcal{T}_f) = (p, R)$, then (i) in every rewrite derivation from g^n to a constructor-rooted term-graph, a descendant of g^p is rewritten at the root, in one or more steps, into a constructor-rooted term-graph; (ii) g^p is a redex of g matched by the left-hand side of R; (iii) g^p is an outermost redex of g^n; (iv) if $\varphi(g^n, \mathcal{T})$ is not defined, then g^n cannot be rewritten into a constructor-rooted term-graph.*

Proposition 4.8 *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an inductively sequential tGRS, and g^n a rooted admissible term-graph. If $\Phi(g) = (p, R)$, then g^p is an outermost needed redex of g^n and g^n can be rewritten at node p with rule R. If $\Phi(g)$ is not defined, then g^n cannot be rewritten into a constructor term-graph.*

Proposition 4.9 *The strategy Φ is c-hyper-normalizing strategy (and thus c-normalizing).*

Graph rewriting does not duplicate data. Thus the number of rewrite steps which are necessary to compute a constructor normal form may be optimized. We obtain the following result for the strategy Φ.

Theorem 4.10 *Let g be an admissible graph and c a constructor graph such that there exists a rewrite derivation $g \xrightarrow{*} c$. Then there exists a constructor graph c' with $c' \sim c$ such that the length of the Φ-derivation $g \xrightarrow{*}_{\Phi} c'$ is less (or equal) to the length of the derivation $g \xrightarrow{*} c$.*

5 Conclusion

We characterized the class of admissible term-graphs. For this class, we showed that the property of confluence of (cyclic) term-graph rewriting relation, induced by weakly orthogonal constructor-based tGRSs, can be recovered, even in the presence of collapsing rules. This result leads us to discover an efficient rewrite strategy for inductively sequential term-graph rewrite systems. This strategy, which has been defined precisely, is c-normalizing and optimal for the class of admissible term-graphs. In [21], a lazy graph rewriting strategy close to ours is described, namely the annotated functional strategy, which combines the discriminating position strategy [24] and rewriting with priority [4]. To our knowledge, no formal result has been proved regarding this strategy.

Our strategy has been successfully extended to parallel rewriting in presence of weakly orthogonal rewrite systems [10]. Its has also been extended in order to develop needed narrowing steps [11,9]. Recently, the framework presented in this paper has been extended in order to deal with non deterministic functions in declarative (functional and logic) languages [3,2]. We are currently investigating new extensions of the presented strategy in a more general setting like term-graphs with priority [12].

References

[1] S. Antoy. Definitional trees. In *Proc. of the 4th Intl. Conf. on Algebraic and Logic programming*, pages 143–157. Springer Verlag LNCS 632, 1992.

[2] S. Antoy and B. Brassel. Computing with subspaces. In *Proc. of the 9th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP07)*, pages 121–130. ACM, 2007.

[3] S. Antoy, D. W. Brown, and S.-H. Chiang. Lazy context cloning for non-deterministic graph rewriting. *Electr. Notes Theor. Comput. Sci.*, 176(1):3–23, 2007.

[4] J. Baeten, J. Bergstra, and J. Klop. Term rewriting systems with priorities. In *Proc. of Conference on Rewriting Techniques and Applications*, pages 83–94, Bordeaux, 1987. Springer Verlag LNCS 256.

[5] H. Barendregt, M. van Eekelen, J. Glauert, R. Kenneway, M. J. Plasmeijer, and M. Sleep. Term graph rewriting. In *PARLE'87*, pages 141–158. Springer Verlag LNCS 259, 1987.

[6] H. P. Barendregt, M. C. J. D. van Eekelen, M. J. Plasmeijer, J. R. W. Glauert, R. Kennaway, and M. R. Sleep. Lean: an intermediate language based on graph rewriting. *Parallel Computing*, 9(2):163–177, 1989.

[7] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic approaches to graph transformation - part I: Basic concepts and double pushout approach. In *Handbook of Graph Grammars*, pages 163–246, 1997.

[8] D. Duval, R. Echahed, and F. Prost. Modeling pointer redirection as cyclic term-graph rewriting. *Electr. Notes Theor. Comput. Sci.*, 176(1):65–84, 2007.

[9] R. Echahed and J. C. Janodet. Admissible graph rewriting and narrowing. In *Proc. of Joint International Conference and Symposium on Logic Programming (JICSLP'98)*, pages 325–340. MIT Press, June 1998.

[10] R. Echahed and J.-C. Janodet. Parallel admissible graph rewriting. In *Recent Trends in Algebraic Development Techniques, 13th International Workshop, WADT '98, Lisbon, Portugal, April 2-4, 1998, Selected Papers*, volume 1589 of *Lecture Notes in Computer Science*, pages 122–137. Springer, 1999.

[11] R. Echahed and J.-C. Janodet. Completeness of admissible-graph collapsing narrowing. In *Proceedings of the Joint APPLIGRAPH/GETGRATS Workshop on Graph Transformation Systems (GRATRA)*, pages 123–132, Berlin, March 2000.

[12] R. Echahed and N. Peltier. Non strict confluent rewrite systems for data-structures with pointers. In *Term Rewriting and Applications, 18th International Conference, RTA 2007, Paris, France, June 26-28, 2007, Proceedings*, volume 4533 of *Lecture Notes in Computer Science*, pages 137–152. Springer, 2007.

[13] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 2: Applications, Languages and Tools*. World Scientific, 1999.

[14] H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 3: Concurrency, Parallelism and Distribution*. World Scientific, 1999.

[15] A. Habel, J. Muller, and D. Plump. Double-pushout graph transformation revisited. *Mathematical Structures in Computer Science*, 11, 2001.

[16] G. Huet and J.-J. Lévy. Computations in orthogonal term rewriting systems. In J.-L. Lassez and G. Plotkin, editors, *Computational logic: essays in honour of Alan Robinson*. MIT Press, Cambridge, MA, 1991. Previous version: Call by need computations in non-ambiguous linear term rewriting systems, Technical Report 359, INRIA, Le Chesnay, France, 1979.

[17] J. R. Kennaway, J. K. Klop, M. R. Sleep, and F. J. D. Vries. On the adequacy of graph rewriting for simulating term rewriting. *ACM Transactions on Programming Languages and Systems*, 16(3):493–523, 1994. Previous version: Technical Report CS-R9204, CWI, Amsterdam, 1992.

[18] J. R. Kennaway, J. K. Klop, M. R. Sleep, and F. J. D. Vries. Transfinite reduction in orthogonal term rewriting systems. *Information and Computation*, pages 18–38, 1995.

[19] R. Kennaway. Implementing term rewrite languages in dactl. *Theor. Comput. Sci.*, 72(2&3):225–249, 1990.

[20] J. W. Klop and A. Middeldörp. Sequentiality in orthogonal term rewriting systems. *Journal of Symbolic Computation*, pages 161–195, 1991.

[21] P. Koopman, J. Smetsers, M. van Eekelen, and M. Plasmeijer. Graph rewriting using the annotated functional strategy. In R. Sleep, R. Plasmeijer, and M. van Eekelen, editors, *Term Graph Rewriting: Theory and Practice*, chapter 23, pages 317–332. John Wiley, New York, 1993.

[22] M. J. O'Donnell. *Computing in Systems Described by Equations*. Springer Verlag LNCS 58, 1977.

[23] M. J. Plasmeijer. Clean : a programming environment based on term graph rewriting. Technical report, 1995. On the Web : http://www.elsevier.nl/locate/tcs.

[24] R. Plasmeijer and M. van Eekelen. *Functional Programming and Parallel Graph Rewriting*. Addison-Wesley, 1993.

[25] D. Plump. Term graph rewriting. In H. Ehrig, G. Engels, H. J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2, pages 3–61. World Scientific, 1999.

[26] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.

[27] Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.